Joe (Home)

Network Working Group
Request for Comments: 913

Mark K. Lottor
MIT
September 1984

# Simple File Transfer Protocol

## STATUS OF THIS MEMO

This RFC suggests a proposed protocol for the ARPA-Internet community, and requests discussion and suggestions for improvements. Distribution of this memo is unlimited.

## INTRODUCTION

SFTP is a simple file transfer protocol. It fills the need of people wanting a protocol that is more useful than TFTP but easier to implement (and less powerful) than FTP. SFTP supports user access control, file transfers, directory listing, directory changing, file renaming and deleting.

SFTP can be implemented with any reliable 8-bit byte stream oriented protocol, this document describes its TCP specification. SFTP uses only one TCP connection; whereas TFTP implements a connection over UDP, and FTP uses two TCP connections (one using the TELNET protocol).

## THE PROTOCOL

SFTP is used by opening a TCP connection to the remote hosts' SFTP port (115 decimal). You then send SFTP commands and wait for replies. SFTP commands sent to the remote server are always 4 ASCII letters (of any case) followed by a space, the argument(s), and a <NULL>. The argument can sometimes be null in which case the command is just 4 characters followed by <NULL>. Replies from the server are always a response character followed immediately by an ASCII message string terminated by a <NULL>. A reply can also be just a response character and a <NULL>.

```
<command> : = <cmd> [<SPACE> <args>] <NULL>

<cmd> : =  USER ! ACCT ! PASS ! TYPE ! LIST ! CDIR
           KILL ! NAME ! DONE ! RETR ! STOR

<response> : = <response-code> [<message>] <NULL>

<response-code> : =  + | - |   | !

<message> can contain <CRLF>
```

Commands that can be sent to the server are listed below. The server

replies to each command with one of the possible response codes listed under each message. Along with the response, the server should optionally return a message explaining the error in more detail. Example message texts are listed but do not have to be followed. All characters used in messages are ASCII 7-bit with the high-order bit zero, in an 8 bit field.

The response codes and their meanings:

+ Success.

- Error.

  An error occurred while processing your command.

  Number.

  The number-sign is followed immediately by ASCII digits representing a decimal number.

! Logged in.

  You have sent enough information to be able to log yourself in. This is also used to mean you have sent enough information to connect to a directory.

To use SFTP you first open a connection to the remote SFTP server. The server replies by sending either a positive or negative greeting, such as:

+MIT-XX SFTP Service

  (the first word should be the host name)

-MIT-XX Out to Lunch

If the server send back a '-' response it will also close the connection, otherwise you must now send a USER command.

USER user-id

Your userid on the remote system.

The reply to this command will be one of:

!<user-id> logged in

Meaning you don't need an account or password or you specified a user-id not needing them.

+User-id valid, send account and password

-Invalid user-id, try again

If the remote system does not have user-id's then you should send an identification such as your personal name or host name as the argument, and the remote system would reply with '+'.

ACCT account

The account you want to use (usually used for billing) on the remote system.

Valid replies are:

! Account valid, logged-in

Account was ok or not needed. Skip the password.

+Account valid, send password

Account ok or not needed. Send your password next.

-Invalid account, try again

PASS password

Your password on the remote system.

Valid replies are:

! Logged in

Password is ok and you can begin file transfers.

+Send account

Password ok but you haven't specified the account.

-Wrong password, try again

You cannot specify any of the following commands until you receive a '!' response from the remote system.

TYPE { A | B | C }

The mapping of the stored file to the transmission byte stream is controlled by the type. The default is binary if the type is not specified.

A - ASCII

The ASCII bytes are taken from the file in the source system, transmitted over the connection, and stored in the file in the destination system.

The data is the 7-bit ASCII codes, transmitted in the low-order 7 bits of 8-bit bytes. The high-order bit of the transmission byte must be zero, and need not be stored in the file.

The data is "NETASCII" and is to follow the same rules as data sent on Telnet connections. The key requirement here is that the local end of line is to be converted to the pair of ASCII characters CR and LF when transmitted on the connection.

For example, TOPS-20 machines have 36-bit words. On TOPS-20 machines, The standard way of labeling the bits is 0 through 35 from high-order to low-order. On TOPS-20 the normal way of storing ASCII data is to use 5 7-bit bytes per word. In ASCII mode, the bytes transmitted would be [0-6], [7-13], [14-20], [21-27], [28-34], (bit 35 would not be transmitted), each of these 7-bit quantities would be transmitted as the low-order 7 bits of an 8-bit byte (with the high-order bit zero).

For example, one disk page of a TOPS-20 file is 512 36-bit words. But using only 35 bits per word for 7-bit bytes, a page is 17920 bits or 2560 bytes.

B - BINARY

The 8-bit bytes are taken from the file in the source system, transmitted over the connection, and stored in the file in the destination system.

The data is in 8-bit units. In systems with word sizes which are not a multiple of 8, some bits of the word will not be transmitted.

For example, TOPS-20 machines have 36-bit words. In binary mode, the bytes transmitted would be [0-7], [8-15], [16-23], [24-31], (bits 32-35 would not be transmitted).

For example, one disk page of a TOPS-20 file is 512 36-bit words. But using only 32 bits per word for 8-bit bytes, a page is 16384 bits or 2048 bytes.

C - CONTINUOUS

The bits are taken from the file in the source system continuously, ignoring word boundaries, and sent over the connection packed into 8-bit bytes. The destination system stores the bits received into the file continuously, ignoring word boundaries.

For systems on machines with a word size that is a multiple of 8 bits, the implementation of binary and continuous modes should be identical.

For example, TOPS-20 machines have 36-bit words. In continuous mode, the bytes transmitted would be [first word, bits 0-7], [first word, bits 8-15], [first word, bits 16-23], [first word, bits 24-31], [first word, bits 32-35 + second word, bits 0-3], [second word, bits 4-11], [second word, bits 12-19], [second word, bits 20-27], [second word, bits 28-35], then the pattern repeats.

For example, one disk page of a TOPS-20 file is 512 36-bit words. This is 18432 bits or 2304 8-bit bytes.

Replies are:

+Using { Ascii | Binary | Continuous } mode

-Type not valid

LIST { F | V } directory-path

A null directory-path will return the current connected directory listing.

F specifies a standard formatted directory listing.

An error reply should be a '-' followed by the error message from the remote systems directory command. A directory listing is a '+' followed immediately by the current directory path specification and a <CRLF>. Following the directory path is a single line for each file in the directory. Each line is just the file name followed by <CRLF>. The listing is terminated with a <NULL> after the last <CRLF>.

V specifies a verbose directory listing.

An error returns '-' as above. A verbose directory listing is a '+' followed immediately by the current directory path specification and a <CRLF>. It is then followed by one line per file in the directory (a line ending in <CRLF>). The line returned for each file can be of any format. Possible information to return would be the file name, size, protection, last write date, and name of last writer.

CDIR new-directory

This will change the current working directory on the remote host to the argument passed.

Replies are:

!Changed working dir to <new-directory>

-Can't connect to directory because: (reason)

+directory ok, send account/password

If the server replies with '+' you should then send an ACCT or PASS command. The server will wait for ACCT or PASS commands until it returns a '-' or '!' response.

Replies to ACCT could be:

!Changed working dir to <new-directory>

+account ok, send password

-invalid account

Replies to PASS could be:

!Changed working dir to <new-directory>

+password ok, send account

-invalid password

KILL file-spec

This will delete the file from the remote system.

Replies are:

+<file-spec> deleted

-Not deleted because (reason)

NAME old-file-spec

Renames the old-file-spec to be new-file-spec on the remote system.

Replies:

+File exists

-Can't find <old-file-spec>

NAME command is aborted, don't send TOBE.

If you receive a '+' you then send:

TOBE new-file-spec

The server replies with:

+<old-file-spec> renamed to <new-file-spec>

-File wasn't renamed because (reason)

DONE

Tells the remote system you are done.

The remote system replies:

+(the message may be charge/accounting info)

and then both systems close the connection.

RETR file-spec

Requests that the remote system send the specified file.

Receiving a '-' from the server should abort the RETR command and the server will wait for another command.

The reply from the remote system is:

<number-of-bytes-that-will-be-sent> (as ascii digits)

-File doesn't exist

You then reply to the remote system with:

SEND (ok, waiting for file)

The file is then sent as a stream of exactly the number of 8-bit bytes specified. When all bytes are received control passes back to you (the remote system is waiting for the next command). If you don't receive a byte within a reasonable amount of time you should abort the file transfer by closing the connection.

STOP (You don't have enough space to store file)

Replies could be:

+ok, RETR aborted

You are then ready to send another command to the remote host.

STOR { NEW | OLD | APP } file-spec

Tells the remote system to receive the following file and save it under that name.

Receiving a '-' should abort the STOR command sequence and the server should wait for the next command.

NEW specifies it should create a new generation of the file and not delete the existing one.

Replies could be:

+File exists, will create new generation of file

+File does not exist, will create new file

-File exists, but system doesn't support generations

OLD specifies it should write over the existing file, if any, or else create a new file with the specified name.

Replies could be:

+Will write over old file

+Will create new file

(OLD should always return a '+')

APP specifies that what you send should be appended to the file on the remote site. If the file doesn't exist it will be created.

Replies could be:

+Will append to file

+Will create file

(APP should always return a '+')

You then send:

SIZE <number-of-bytes-in-file> (as ASCII digits)

where number-of-bytes-in-file

is the exact number of 8-bit bytes you will be sending.

The remote system replies:

+ok, waiting for file

You then send the file as exactly the number of bytes specified above.

When you are done the remote system should reply:

+Saved <file-spec>

-Couldn't save because (reason)

-Not enough room, don't send it

This aborts the STOR sequence, the server is waiting for your next command.

You are then ready to send another command to the remote host.

AN EXAMPLE

An example file transfer. 'S' is the sender, the user process. 'R' is the reply from the remote server. Remember all server replies are terminated with <NULL>. If the reply is more than one line each line ends with a <CRLF>.

```
R: (listening for connection)
S: (opens connection to R)
R: +MIT-XX SFTP Service
S: USER MKL
R: +MKL ok, send password
S: PASS foo
R: ! MKL logged in
S: LIST F PS: <MKL>
R: +PS: <MKL>
   Small.File
   Large.File
S: LIST V
R: +PS: <MKL>
   Small.File  1         69(7)   P775240  2-Aug-84 20:08  MKL
   Large.File  100  255999(8)   P770000  9-Dec-84 06:04  MKL
S: RETR SMALL.FILE
R:  69
S: SEND
R: This is a small file, the file is sent without
   a terminating null.
S: DONE
R: +MIT-XX closing connection
```

EDITORS NOTE

Mark Lotter receives full credit for all the good ideas in this memo. As RFC editor, i have made an number of format changes, a few wording changes, and one or two technical changes (mostly in the TYPEs). I accept full responsibility for any flaws i may have introduced.

A draft form of this memo was circulated for comments. I will attempt to list the issues raised and summarize the pros and cons, and resolution for each.

ASCII Commands vs Binary Operation Codes

The ASCII command style is easier to debug, the extra programming cost in minimal, the extra transmission cost is trivial.

Binary operation codes are more efficient, and a few days of debugging should not out weigh years of use.

Resolution: I have kept the ASCII Commands.

Additional Modes

Pro: For some machines you can't send all the bits in a word using this protocol. There should be some additional mode to allow it.

Con: Forget it, this is supposed to be SIMPLE file transfer. If you need those complex modes use real FTP.

Resolution: I have added the Continuous mode.

CRLF Conversion

Pro: In ASCII type, convert the local end of line indicator to CRLF on the way out of the host and onto the network.

Con: If you require that you have to look at the bytes as you send them, otherwise you can just send them. Most of the time both sides will have the same end of line convention anyway. If someone needs a conversion it can be done with a TECO macro separately.

Resolution: I have required CRLF conversion in ASCII type. If you have the same kind of machines and the same end of line convention you can avoid the extra cost of conversion by using the binary or continuous type.

TCP Urgent

Pro: Use TCP Urgent to abort a transfer, instead of aborting the connection. Then one could retry the file, or try a different file without having to login again.

Con: That would couple SFTP to TCP too much. SFTP is supposed to be able to be work over any reliable 8-bit data stream.

Resolution: I have not made use of TCP Urgent.

Random Access

Pro: Wouldn't it be nice if (WIBNIF) SFTP had a way of accessing parts of a file?

Con: Forget it, this is supposed to be SIMPLE file transfer. If you need random access use real FTP (oops, real FTP doesn't have random access either -- invent another protocol?).

Resolution: I have not made any provision for Random Access.

-- jon postel.